EITF20: Computer Architecture

Part3.1.1: Pipeline - 2

Liang Liu
liang.liu@eit.lth.se

Outline

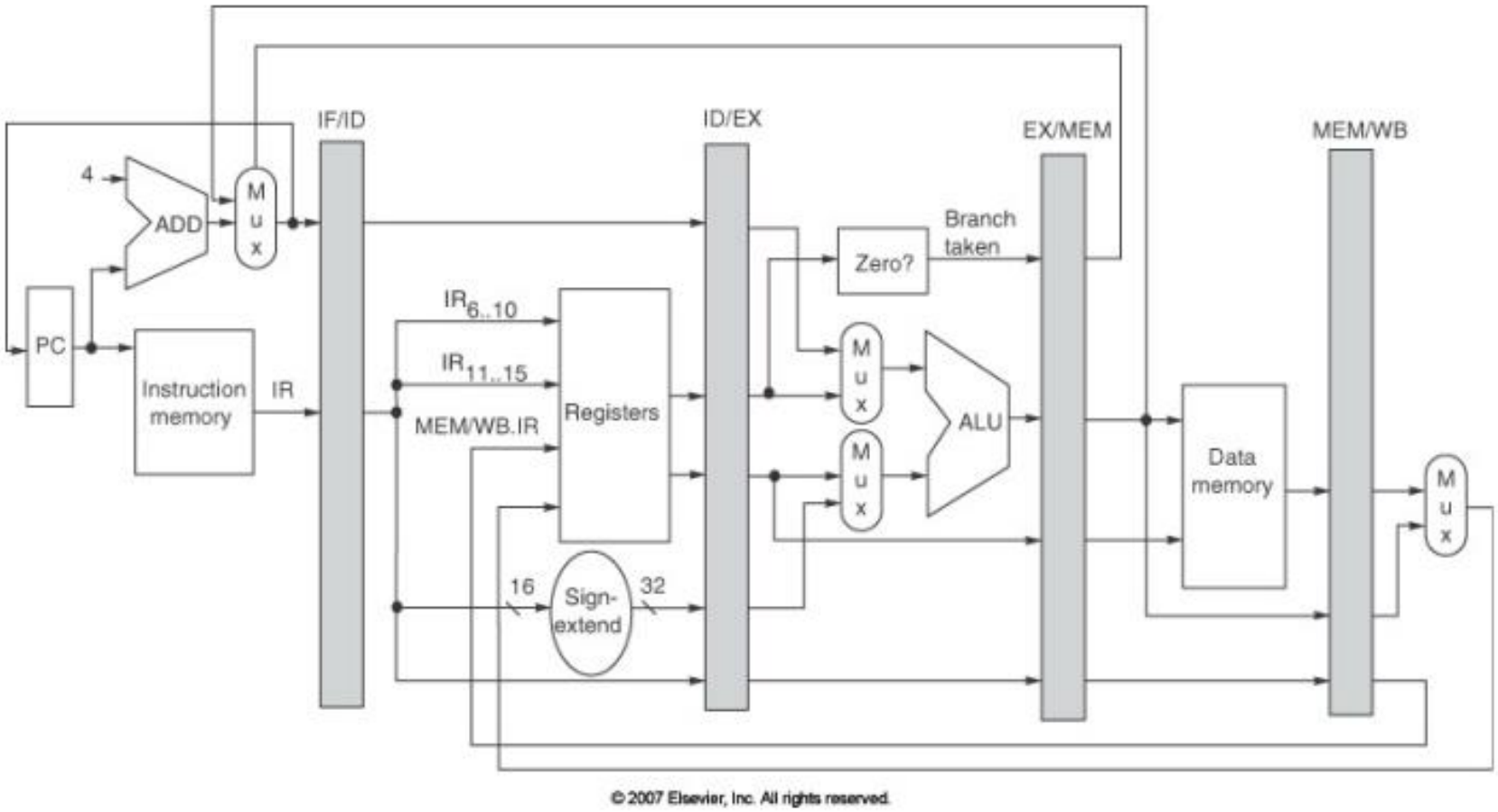
- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- Instruction Scheduling
- Scoreboard

Previous lecture

Introduction to pipeline basics

- General principles of pipelining
- Techniques to avoid pipeline stalls due to hazards
- What makes pipelining hard to implement?
- **Support of multi-cycle instructions in a pipeline**

A pipelined MIPS data-path



Pipeline factors

- **Pipelining doesn't help latency of a single instruction**
- **it helps throughput of the entire workload**
- **Pipeline rate is limited by the slowest pipeline stage**
- **Multiple instructions are executing simultaneously**
- **Potential speedup = Number of pipe stages**
  - Unbalanced lengths of pipe stages reduces speedup
  - Time to fill pipeline and time to drain reduces speedup
  - Hazards reduces speedup

Summary

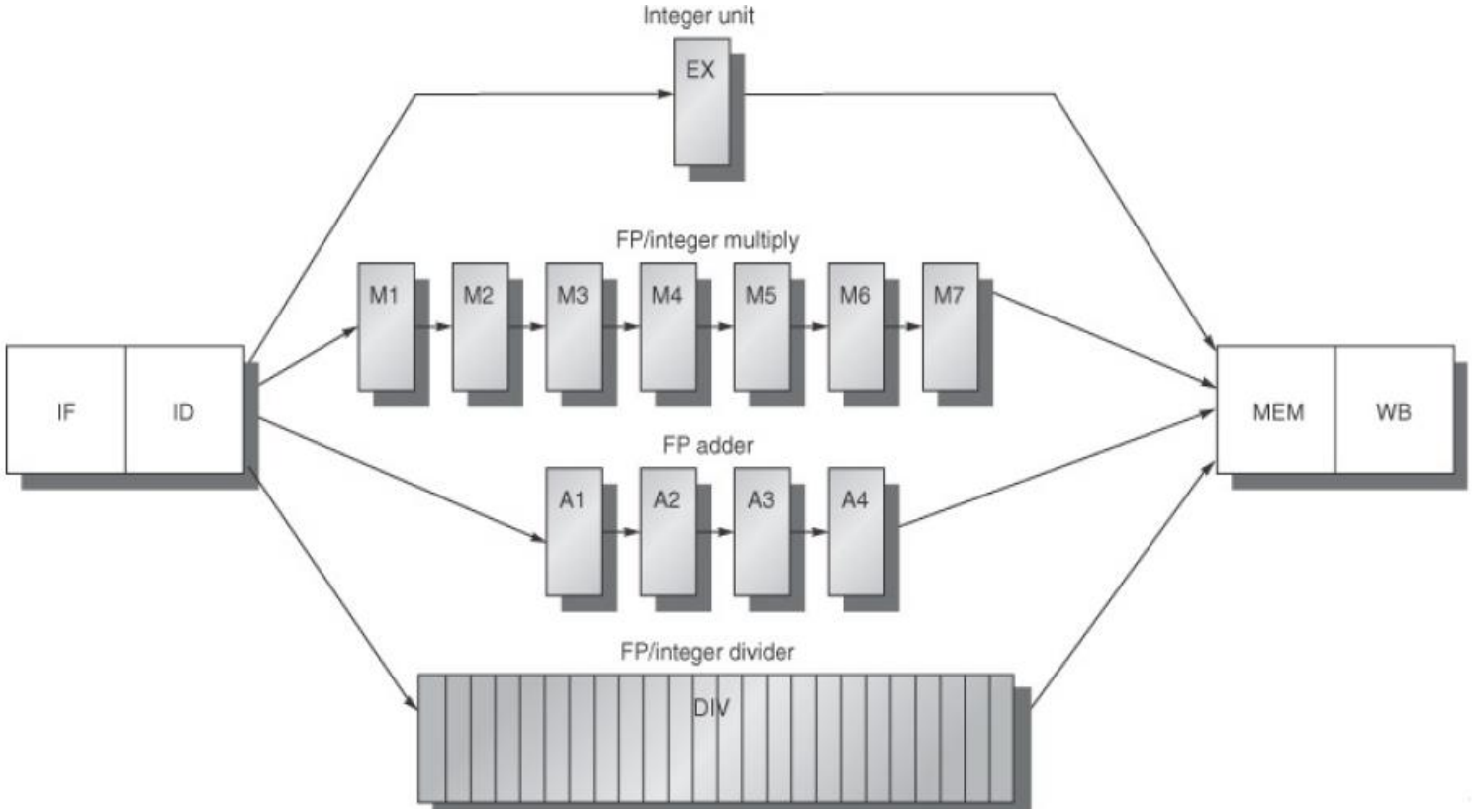
- **Pipelining (ILP):**
  - Speeds up throughput, not latency
  - Speedup ≤ #stages
- **Hazards limit performance, generate stalls:**
  - Structural: need more HW
  - Data (RAW,WAR,WAW): need forwarding and compiler scheduling
  - Control: delayed branch, branch prediction

$$Speedup = \frac{CPI_{unpipelined}}{\text{Ideal CPI} + \text{\# stall-cycles/instr}} * \frac{T_{C_{unpipelined}}}{T_{C_{pipelined}}}$$

$$\approx \frac{\#stages}{1 + \text{\# stall-cycles/instruction}}$$

- **Complications:**
  - Precise exceptions may be difficult to implement

Multi-cycle instruction in pipeline (FP)



FP Instruction	*Latency*	*Initiation Rate*
Add, Subtract	**3**	**1**
Multiply	**6**	**1**
Divide	**24**	**25**

Parallelism between integer and FP

MUL.D	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM
ADD.D		IF	ID	A1	A2	A3	A4	MEM	WB	
DSUB			IF	ID	EX	MEM	WB			
LD				IF	ID	EX	MEM	WB		

- Instructions are issued in order
- Instructions may be completed out of order

Pipeline hazard

- Structural hazards
- RAW hazards
- WAW harzards
- WAR hazards

Pipeline hazard

- **Structural hazards. Stall in ID stage if:**
  - The functional unit is occupied (applicable to DIV only)
  - Any instruction already executing will reach the MEM/WB stage at the same time as this one
- **RAW hazards:**
  - Normal bypassing from MEM and WB stages
  - Stall in ID stage if any of the source operands is destination operand in any of the FP functional units

	Clock cycle number										
Instruction	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB

Pipeline hazard

- **WAR hazards?**
  - There are no WAR-hazards since the operands are read (in ID) before the EX-stages in the pipeline
- **WAW hazard**

```
DIV.D    F0,F2,F3    FP divide 24 cycles
...
SUB.D    F0,F8,F10   FP subtract 3 cycles
```

- SUB finishes before DIV which will overwrite the result from SUB!
- Are eliminated by stalling SUB until DIV reaches MEM stage
- When WAW hazard is a problem?

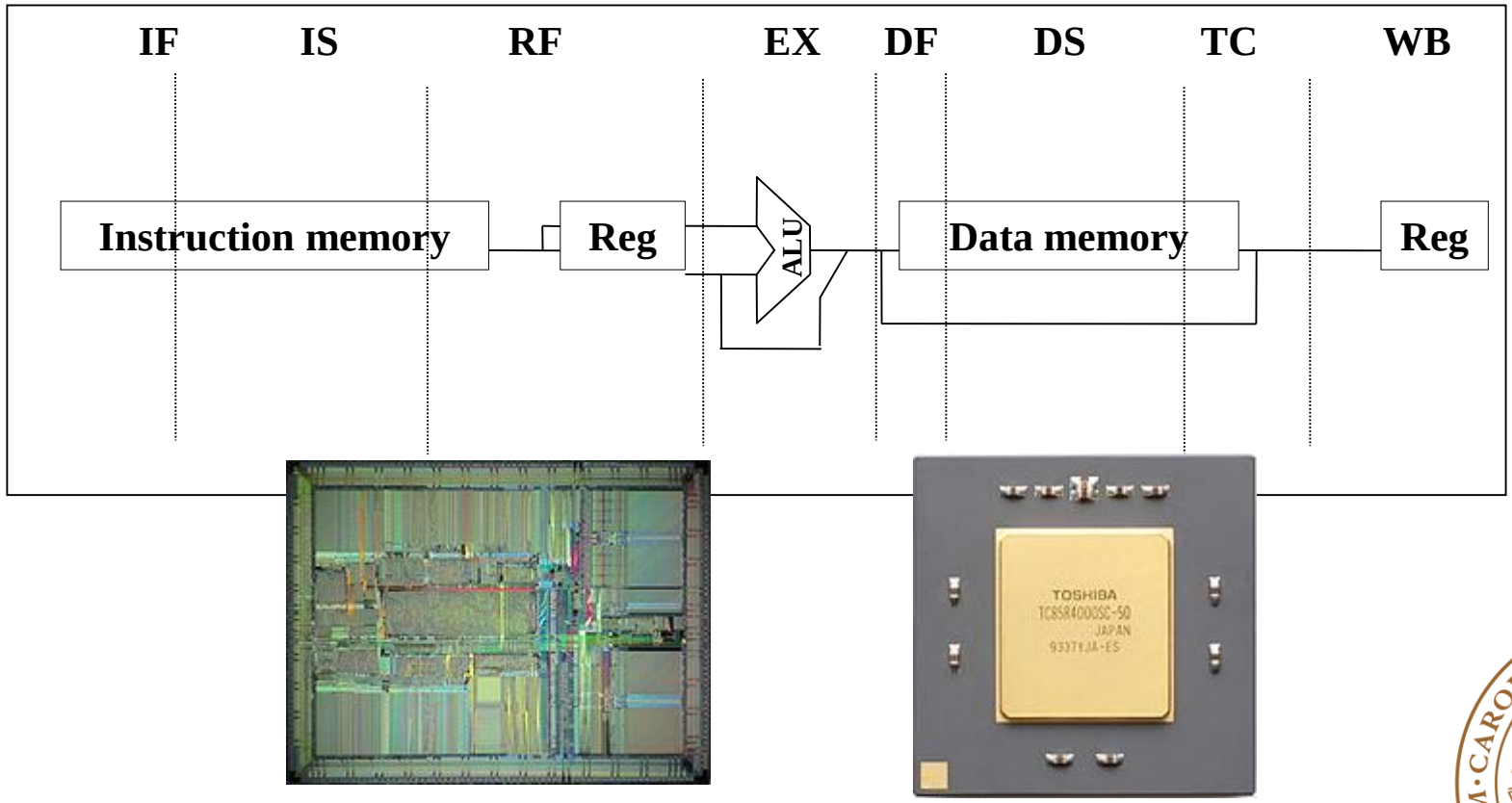
Outline

- Reiteration
- **Case Study: MIPS R4000**
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- Instruction Scheduling
- Scoreboard

The MIPS R4000

R4000 - MIPS64:

- First (1992) true 64-bit architecture (addresses and data)
- Clock frequency (1997): 100 MHz-250 MHz
- Medium deep 8 stage pipeline (super-pipelined)

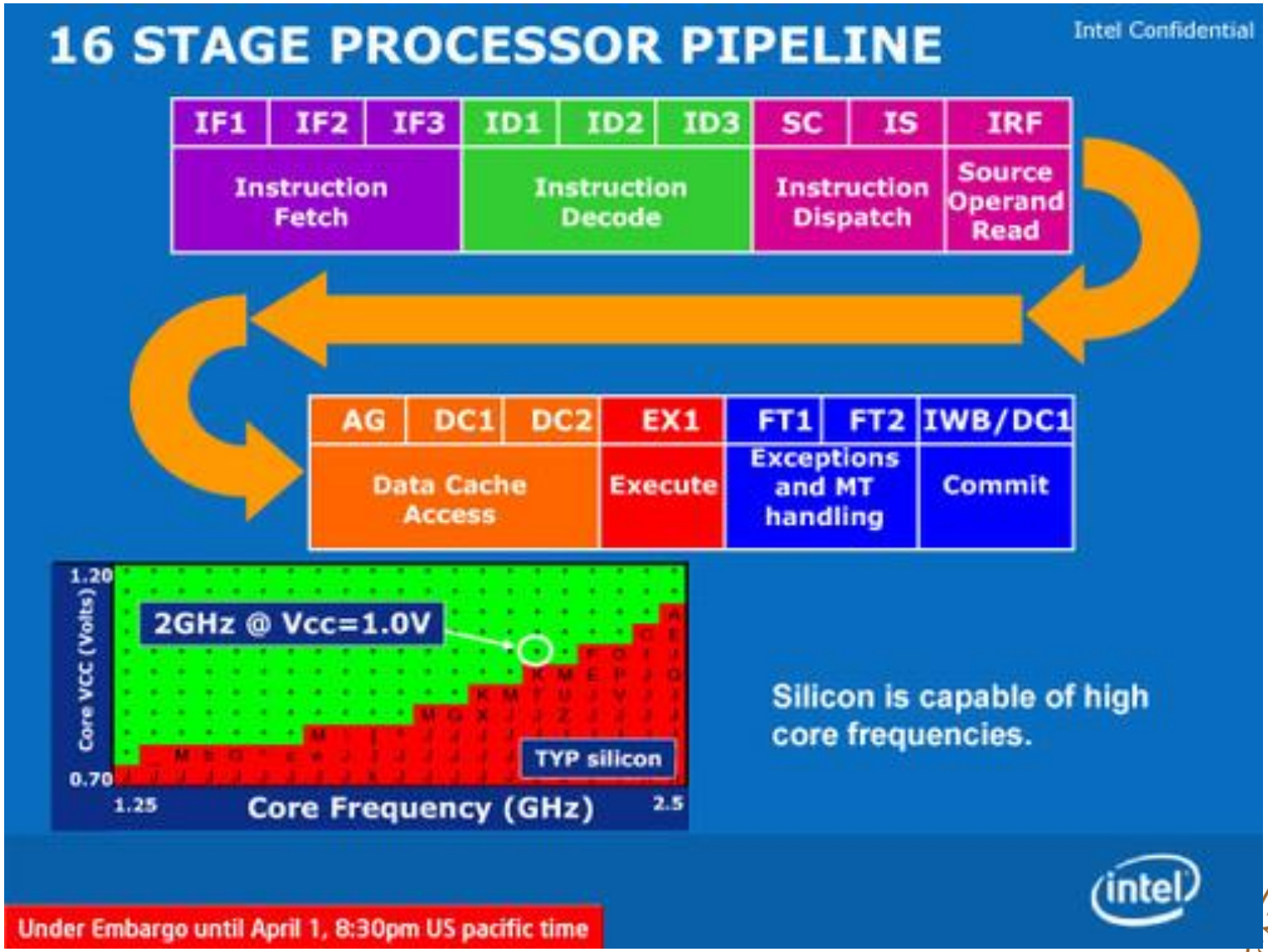


The MIPS R4000

8 Stage Pipeline:

- IF – first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access
- IS – second half of access to instruction cache
- RF – instruction decode and register fetch, hazard checking and also instruction cache hit detection
- EX – execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation
- DF – data fetch, first half of access to data cache
- DS – second half of access to data cache
- TC – tag check, determine whether the data cache access hit
- WB – write back for loads and register-register operations

Modern CPU architecture – Intel Atom



Deeper pipeline

Implications of deeper pipeline

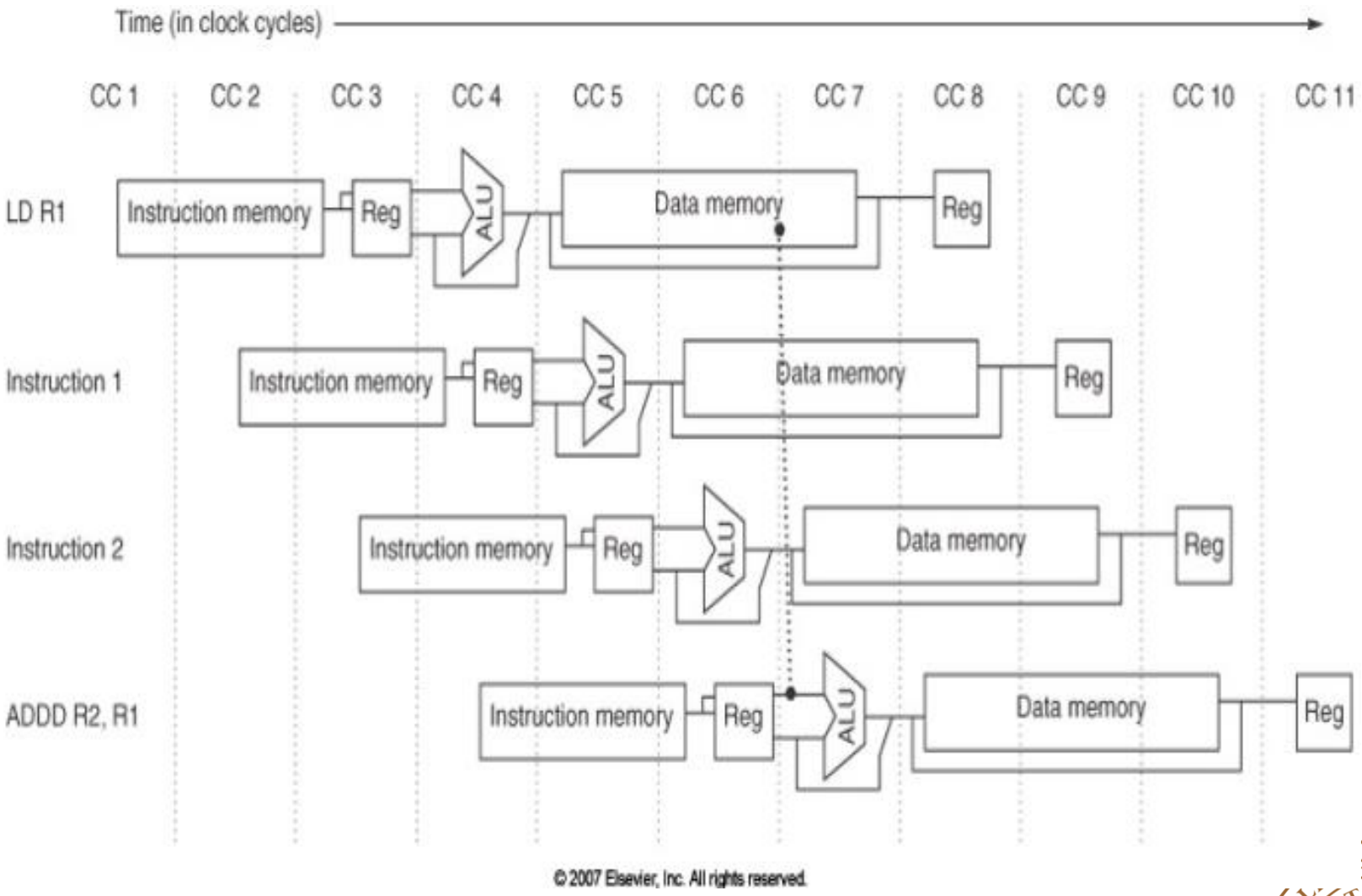
- load latency: 2 cycles
- branch latency: 3 cycles (incl. one delay slot) ⇒ High demands on the compiler
- Bypassing (forwarding) from more stages
- More instructions “in flight” in pipeline
- Faster clock, larger latencies, more stalls

Win or loose?

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

Performance equation: CPI ∗ Tc must be lower for the longer pipeline to make it worthwhile

Load penalties



2 stalls even with forwarding

Load penalties

Instruction	Clock number								
	1	2	3	4	5	6	7	8	9
LD R1,...	IF	IS	RF	EX	DF	DS	TC	WB	
DADD R2,R1,R4		IF	IS	RF	stall	stall	EX	DF	DS
DSUB R3,R1,R5			IF	IS	stall	stall	RF	EX	DF

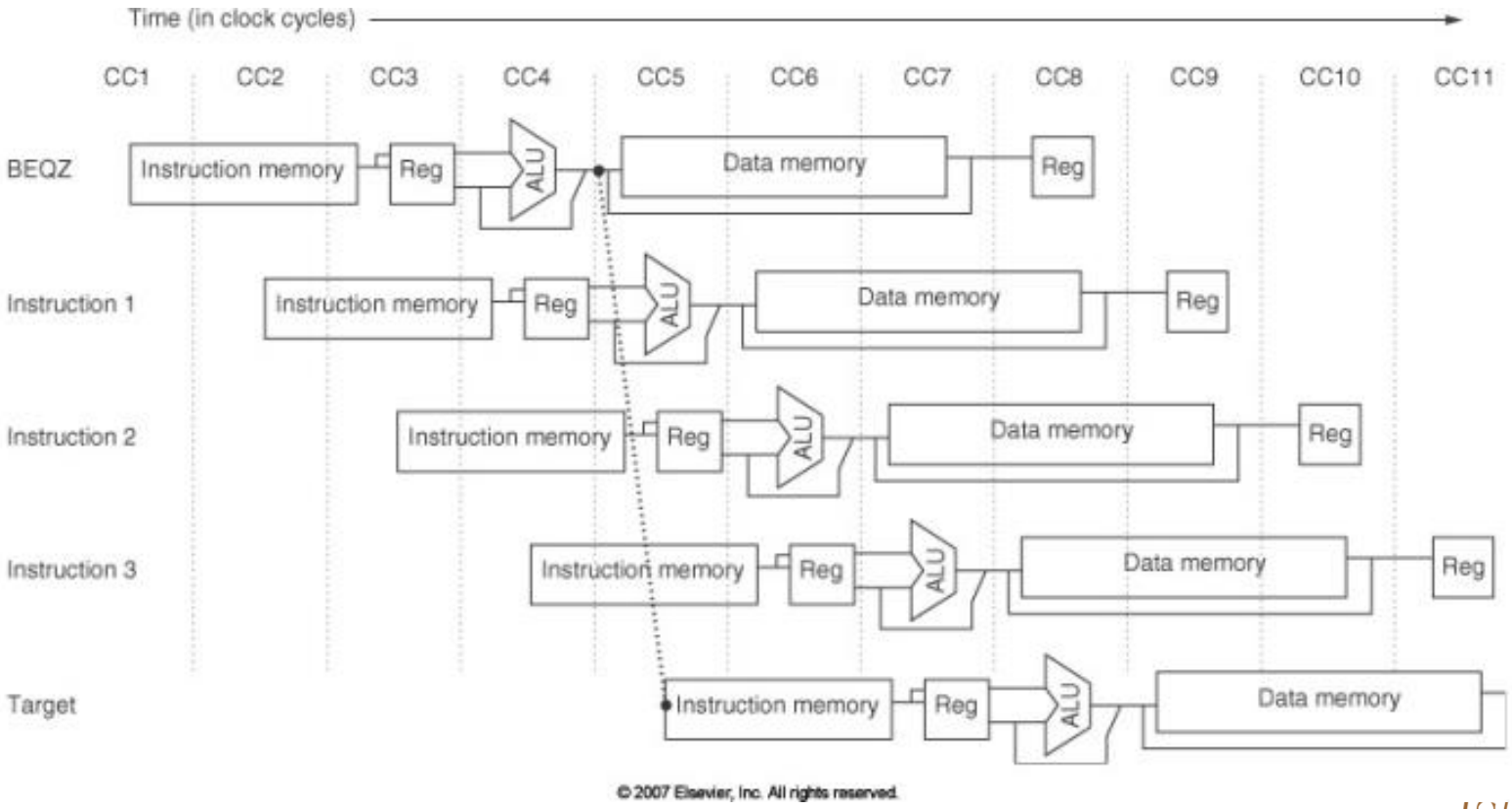
3 load pipeline stages

- Data is available after the DS stage (second stage in 'Data memory')
  ⇒ latency 2 clock cycles
- Four bypasses (EX/DF/DS/TC) instead of two (EX/MEM) because of the two extra pipe stages to read from memory

Branch penalties

Handle branch hazard

- 3 branch delay slot (comparing to simple MIPS)
- Predict-Not taken scheme squashes the next two sequentially fetched instructions if the branch is taken (given on delay slot)

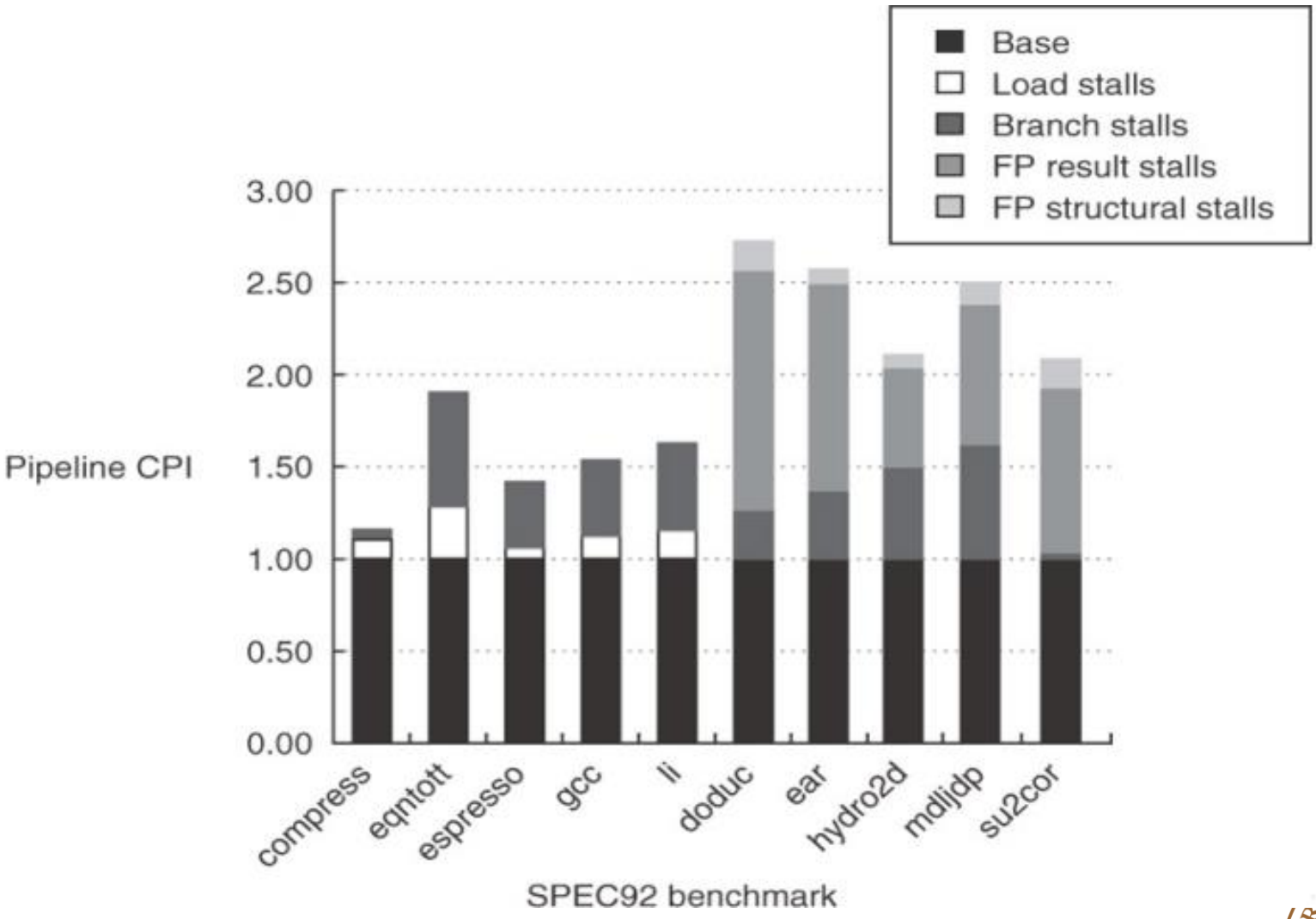


Branch penalties (predict not taken)

branch taken									
Instruction	Clock number								
	1	2	3	4	5	6	7	8	9
Branch instr	IF	IS	RF	EX	DF	DS	TC	WB	
Delay slot		IF	IS	RF	EX	DF	DS	TC	WB
Squash			IF	IS	s	s	s	s	s
Squash				IF	s	s	s	s	s
Branch target					IF	IS	RF	EX	DF

branch not taken									
Instruction	Clock number								
	1	2	3	4	5	6	7	8	9
Branch instr	IF	IS	RF	EX	DF	DS	TC	WB	
Delay slot		IF	IS	RF	EX	DF	DS	TC	WB
Branch instr +2			IF	IS	RF	EX	DF	DS	TC
Branch instr +3				IF	IS	RF	EX	DF	DS

R4000 performance



R4000 performance

	CPI penalties			
Average CPI	Load	Branch	FP data hazard	FP struct hazard
2.00	0.10	0.36	0.46	0.09

(SPEC92 CPI measurements)

R4000 performance

- The penalty for control hazards is very high for integer programs
- The penalty for FP data hazards is also high
- The higher clock frequency compensates for the higher CPI

Outline

- Reiteration
- Case Study: MIPS R4000
- **Instruction Level Parallelism**
- Branch Prediction
- Dependencies
- Instruction Scheduling
- Scoreboard

Instruction level parallelism (ILP)

- **ILP: Overlap execution of unrelated instructions: Pipelining**
- **Two main approaches to increasing the amount of parallelism among instructions:**
  - DYNAMIC ⇒ hardware detects parallelism (Desktop)
  - STATIC ⇒ software detects parallelism (Embedded)
  - Often a mix between both
- **Pipeline CPI = Ideal CPI + Structural stalls + Data hazard stalls + Control stalls**

Instruction level parallelism (ILP)

```
          DIVD   F0,F2,F4
Example:  ADDD   F10,F1,F2
          SUBD   F8,F8,F14
```

- **These instructions are independent and could be executed in parallel**
- **MIPS program has 15-25% average branch frequency:**
  - 5 sequential instructions / branch
  - These 5 instructions may depend on each other
  - Must look beyond a single basic block to get more ILP
- **Loop level parallelism**

Loop-level parallelism

```
for (i=1; i≤1000; i=i+1)
        x[i] = x[i] + 10;
```

- There is very little available parallelism within an iteration
- However, there is parallelism between loop iterations; each iteration is independent of the other

Potential speed up = 1000!

MIPS code for the loop

```
loop:   LD      F0, 0(R1)     ; F0 = array element
        ADDD    F4, F0, F2    ; Add scalar constant
        SD      F4, 0(R1)     ; Save result
        DADDUI  R1, R1, #-8   ; decrement array ptr.
        BNE     R1,R2, loop   ; reiterate if R1 != R2
```

Instruction producing result	Instruction using result	Latency
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0
Integer op	Integer op	0
Integer op	Cond. branch	1

Only consider data hazard

Loop showing stalls

```
1    loop:  LD      F0, 0(R1)      ; F0 = array element
2           stall
3           ADDD    F4, F0, F2     ; Add scalar constant
4           stall
5           stall
6           SD      F4, 0(R1)      ; Save result
7           DADDUI  R1, R1, #-8    ; decrement array ptr.
8           stall
9           BNE     R1,R2 loop     ; reiterate if R1 != R2
```

How can we get rid of these stalls?

Reconstructed loop

```
Loop:   L.D      F0,0(R1)
        DADDUI   R1,R1,#-8
        ADD.D    F4,F0,F2
        stall
        stall
        S.D      F4,8(R1)
        BNE      R1,R2,Loop
```

- Swap DADDUI and SD by changing offset in SD
- 7 clock cycles per iteration
- Sophisticated compiler analysis required

Can we do better?

Unroll loop four times

```
1     loop:  LD      F0, 0(R1)
2            ADDD    F4, F0, F2
3            SD      F4, 0(R1)      ; drop DADDUI & BNE
4            LD      F6, -8(R1)
5            ADDD    F8, F6, F2
6            SD      F8, -8(R1)     ; drop DADDUI & BNE
7            LD      F10, -16(R1)
8            ADDD    F12, F10, F2
9            SD      F12, -16(R1)   ; drop DADDUI & BNE
10           LD      F14, -24(R1)
11           ADDD    F16, F14, F2
12           SD      F16, -24(R1)
13           DADDUI  R1, R1, #-32   ; alter to 4*8
14           BNE     R1, R2, loop
```

- **14 + 4*(1+2) + 1 = 27 clock cycles, or 6.75 per iteration**

Scheduled unrolled loop

```
Loop:   L.D      F0,0(R1)
        L.D      F6,-8(R1)
        L.D      F10,-16(R1)
        L.D      F14,-24(R1)
        ADD.D    F4,F0,F2
        ADD.D    F8,F6,F2
        ADD.D    F12,F10,F2
        ADD.D    F16,F14,F2
        S.D      F4,0(R1)
        S.D      F8,-8(R1)
        DADDUI   R1,R1,#-32
        S.D      F12,16(R1)
        S.D      F16,8(R1)
        BNE      R1,R2,Loop
```

- 14 clock cycles, or 3.5 per iteration

Can we unroll more?

- Code size
- Avaliable registers

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- **Branch Prediction**
- Dependencies
- Instruction Scheduling
- Scoreboard

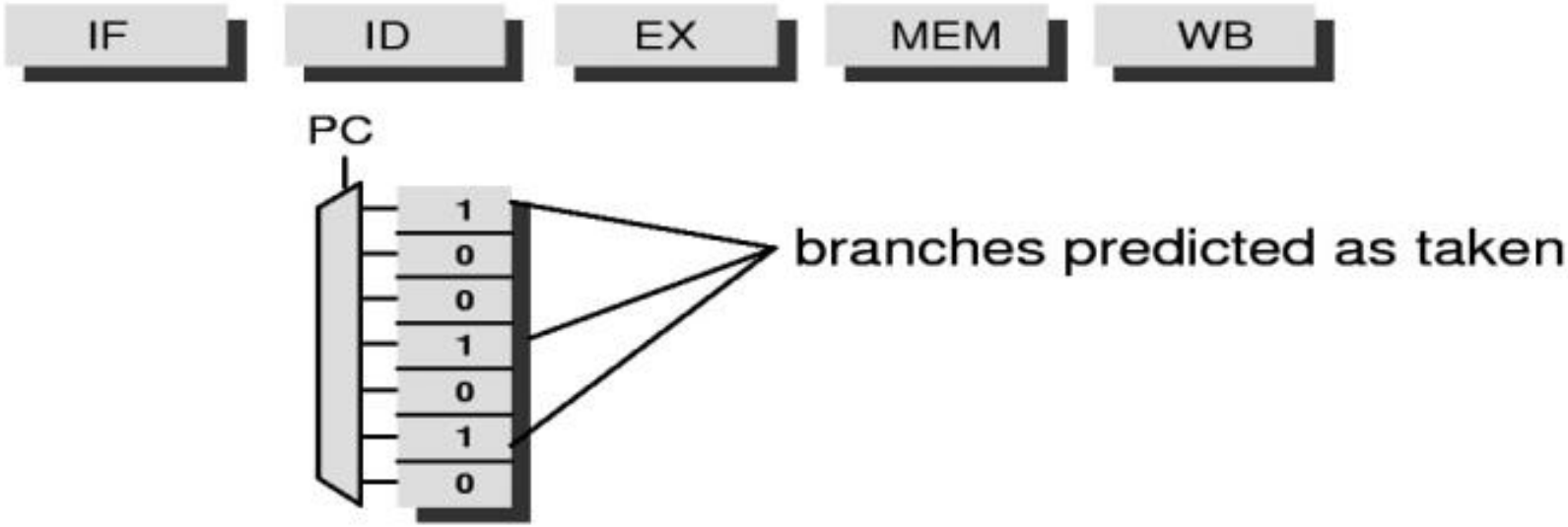
Dynamic branch prediction

- **Branches limit performance because:**
  - Branch penalties
  - Limit to available Instruction Level Parallelism
- **Delayed braches becomes insufficient for deeper pipeline**
- **Dynamic branch prediction to predict the outcome of conditional branches**
- **Benefits:**
  - Reduce the time to when the branch condition is known (if we can predict correctly)
  - Reduce the time to calculate the branch target address (if we can buffer the target address)

Branch history table

Simple branch prediction

- The branch-prediction buffer is indexed by low order part of branch-instruction address (at ID stage)
- The bit corresponding to a branch indicates whether the branch is predicted as taken or not
- When prediction is wrong: invert bit

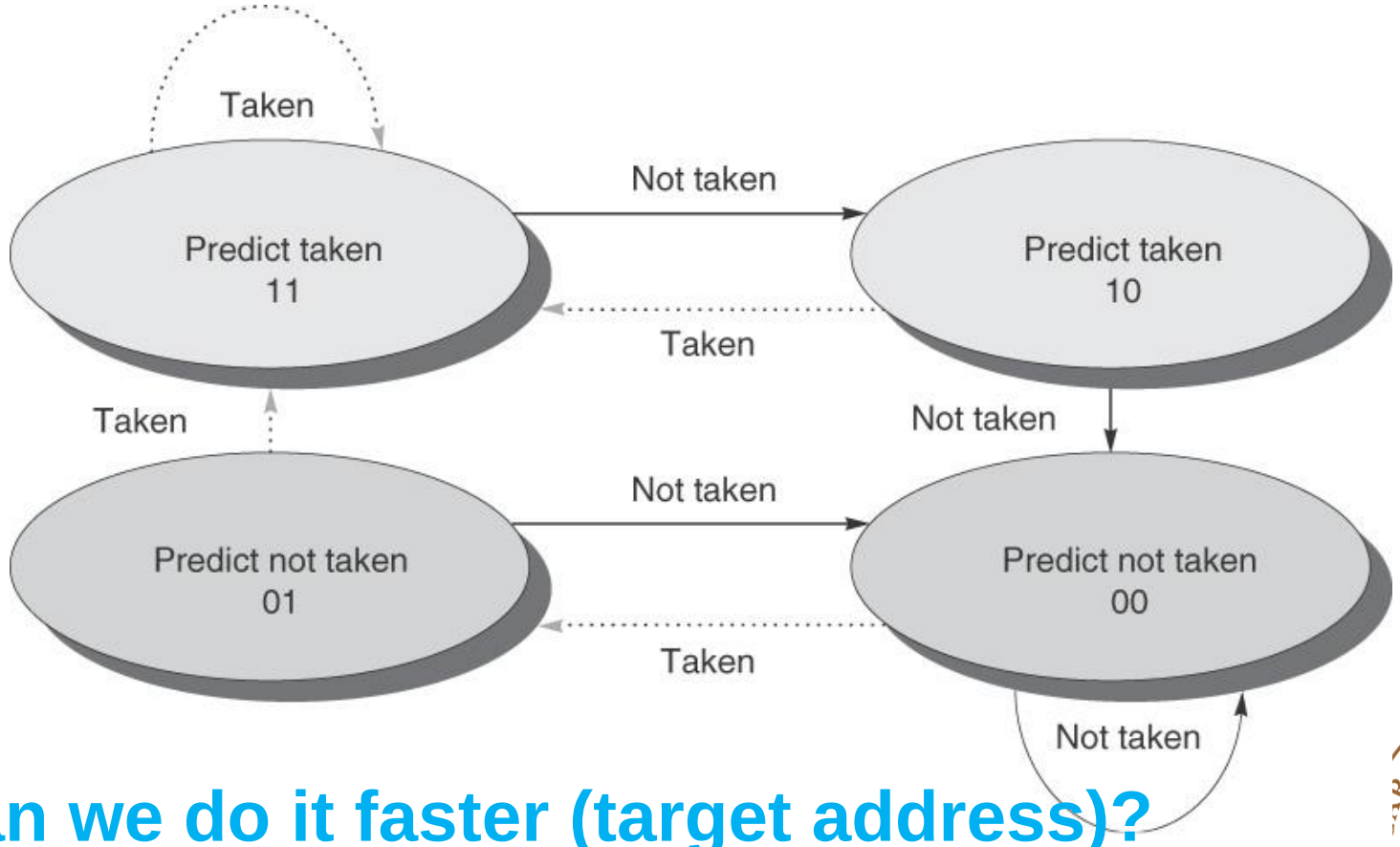


Is one bit good enough?

A 2-bit branch predictor

2-bit branch prediction with saturating counter

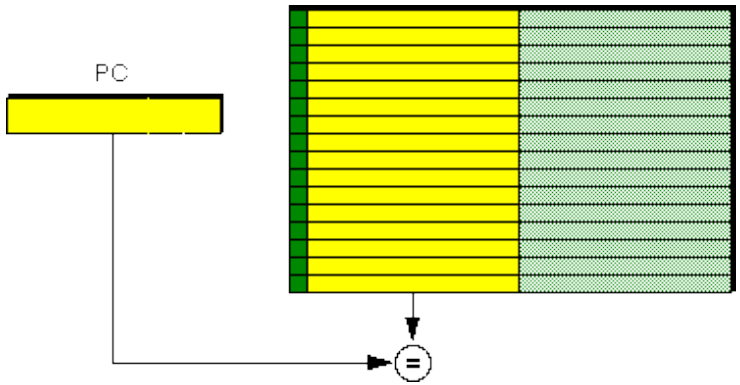
- Requires prediction to miss twice in order to change prediction ⇒ better performance
- 1%-18% miss prediction frequency for SPEC89



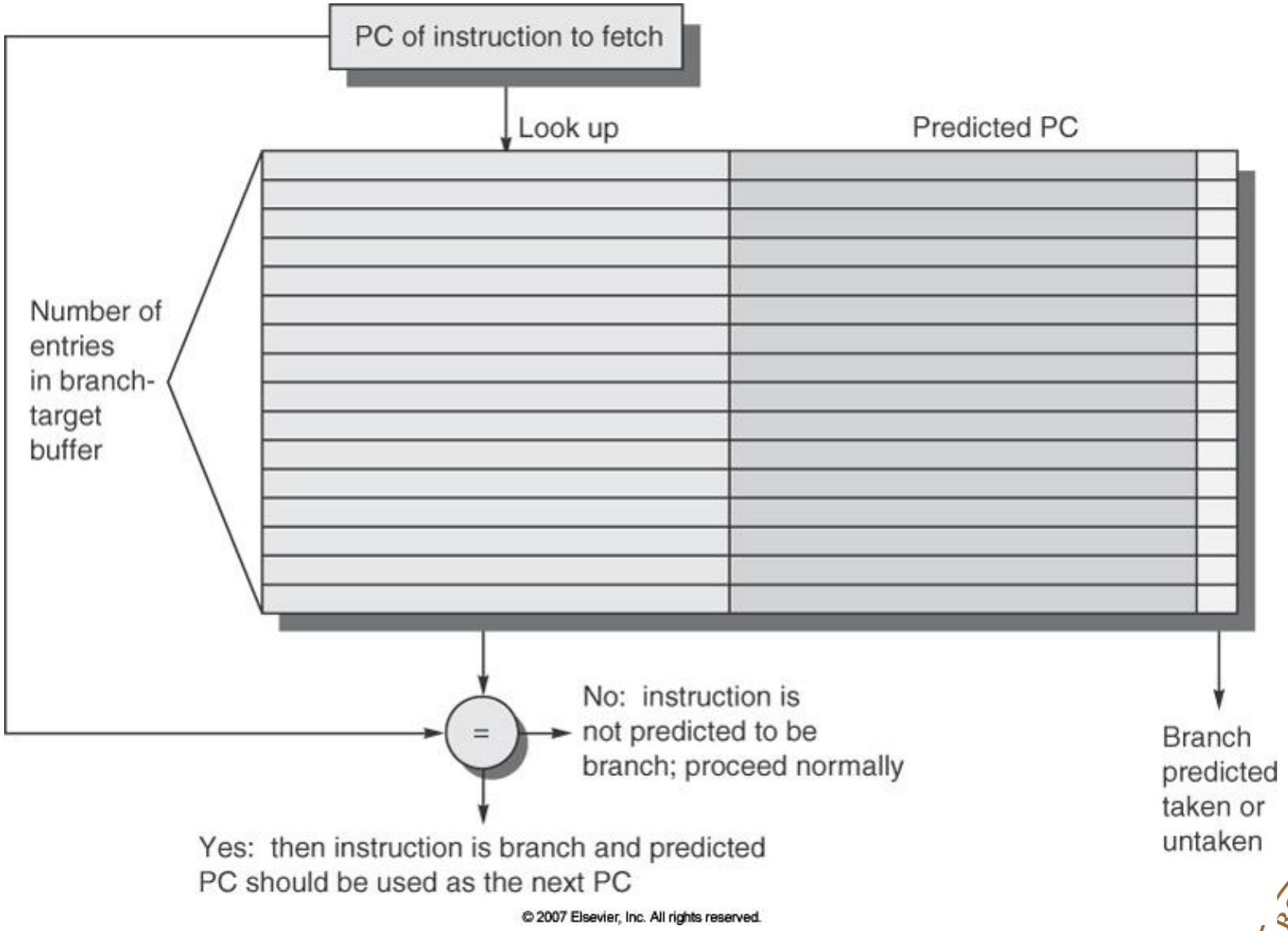
Can we do it faster (target address)?

Branch target buffer

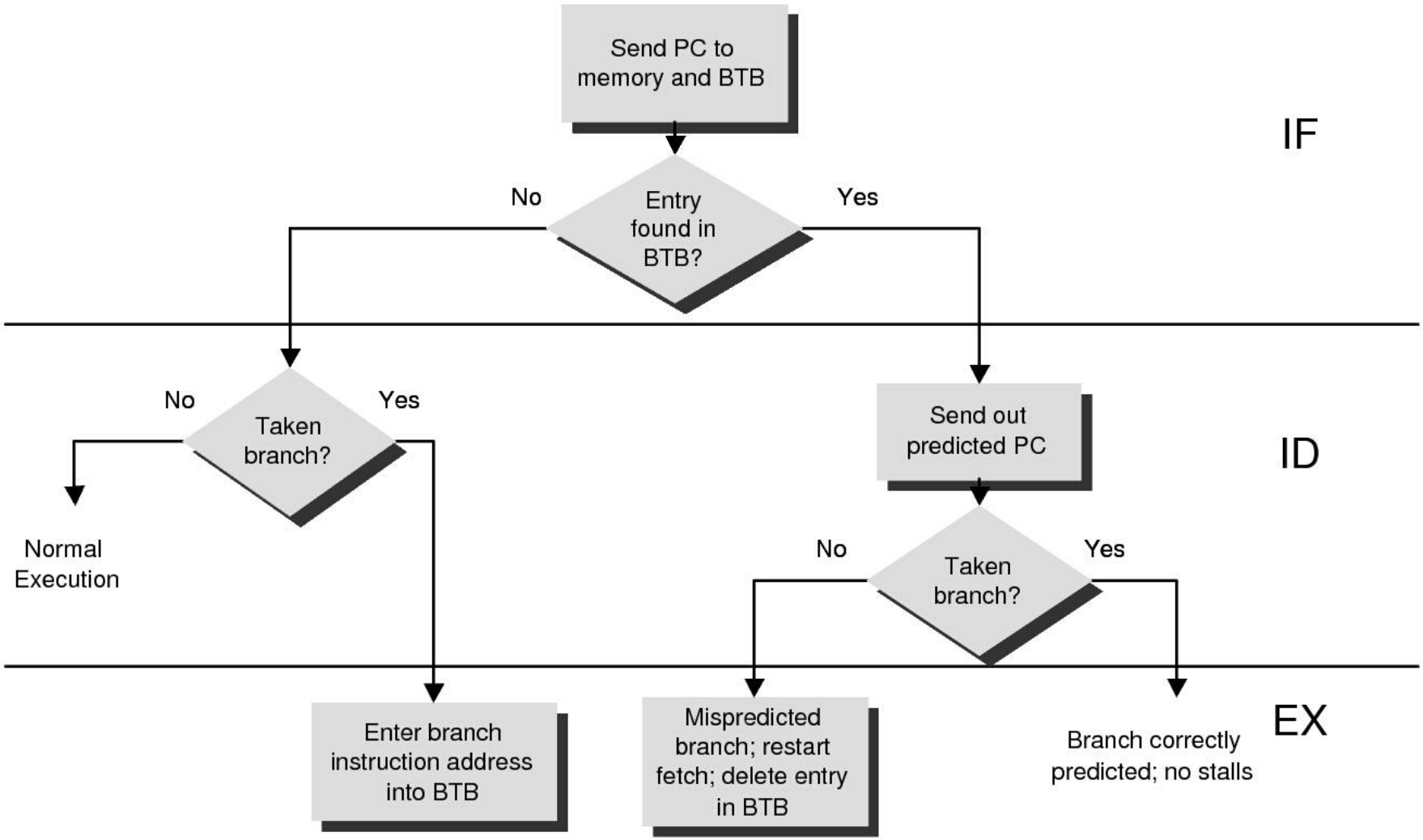
- **Observation: Target address remains the same for a conditional direct branch across dynamic instances**
  - Idea: Store the target address from previous instance and access it with the PC (at **IF** stage)
  - Called Branch Target Buffer (BTB) or Branch Target Address Cache
- **Three steps to be predicted at fetch stage:**
  - Whether the fetched instruction is a branch (somewhat ID)
  - (Conditional) branch direction (only store the predicted-taken braches)
  - Branch target address (if taken)



Branch target buffer



Branch target buffer algorithms



Branch prediction penalties

- **For branch target buffer scheme**

Instruction in buffer	Prediction	Actual branch	Penalty cycles
Yes	Taken	Taken	0
Yes	Taken	Not taken	2
No		Taken	2
No		Not taken	0

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- **Dependencies**
- Instruction Scheduling
- Scoreboard

Dependencies

- **Two instructions must be independent in order to execute in parallel**
- **There are three general types of dependencies that limit parallelism:**
  - Data dependencies (RAW)
  - Name dependencies (WAR,WAW)
  - Control dependencies
- **Dependencies are properties of the program**
- **Whether a dependency leads to a hazard or not is a property of the pipeline implementation**

Data dependencies

- **An instruction *j* is data dependent on instruction *i* if:**
  - Instruction *i* produces a result used by instr. *j*, or
  - Instruction *j* is data dependent on instruction k and instr. k is data dependent on instr. *i*

```
                LD      F0,0(R1)
Example:        ADDD    F4,F0,F2
                SD      0(R1),F4
```

  - Easy to detect for registers

Name dependencies

- **Two instructions use same name (register or memory address) but do not exchange data**
  - Anti-dependence (WAR if hazard in HW)

```
ADDD    F2,F0,F2    ; Must execute before LD
LD      F0,0(R1)
```

  - Output dependence (WAW if hazard in HW)

```
ADDD    F0,F2,F2    ; Must execute before LD
LD      F0,0(R1)
```

Control dependencies

- **Determines order between an instruction and a branch instruction**

Example:

if Test1 then { S1 }

if Test2 then { S2 }

S1 is control dependent on Test1

S2 is control dependent on Test2; but *not* on Test1

- We cannot move an instruction that is dependent on a branch before the branch instruction
- We cannot move an instruction not control dependent on a branch after the branch instruction

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- **Instruction Scheduling**
- Scoreboard

Instruction scheduling

Instruction scheduling

- Scheduling is the process that determines when to **start** a particular instruction, when to **read** its operands, and when to **write** its result,
- Target of scheduling: rearrange instructions to reduce stalls when data or control dependencies are present

Static (compiler at compile-time) scheduling:

- Pro: May look into future; no HW support needed
- Con: Cannot detect all dependencies, esp. across branches; hardware dependent

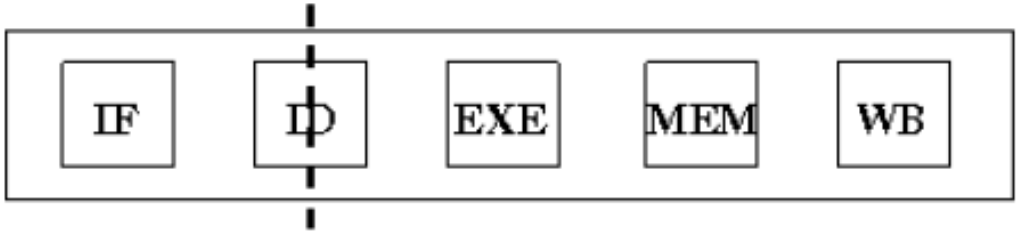
Dynamic (hardware at run-time) scheduling:

- Pro: works when cannot know dependence at compile time; makes the compiler simpler; code for one implementation runs well on another
- Con: Cannot look into the future (practically); HW support needed which complicates the pipeline hardware

Dynamic instruction scheduling

- **Key idea: allow instructions behind stall to proceed**

```
DIVD    F0,F2,F4     ; takes long time
ADDD    F10,F0,F8    ; stalls waiting for F0
```




```
MULTD   F12,F8,F14   ; Let this instr. bypass the ADDD
```

- **MULTD is not data dependent on anything in the pipeline**
- **Enables out-of-order execution ⇒ out-of-order completion**
- **ID stage checks for structural and data dependencies**
  - Scoreboard (CDC 6600 in 1963)
  - Tomasulo (IBM 360/91 in 1967)

Outline

- Reiteration
- Case Study: MIPS R4000
- Instruction Level Parallelism
- Branch Prediction
- Dependencies
- Instruction Scheduling
- **Scoreboard**

Scoreboard pipeline

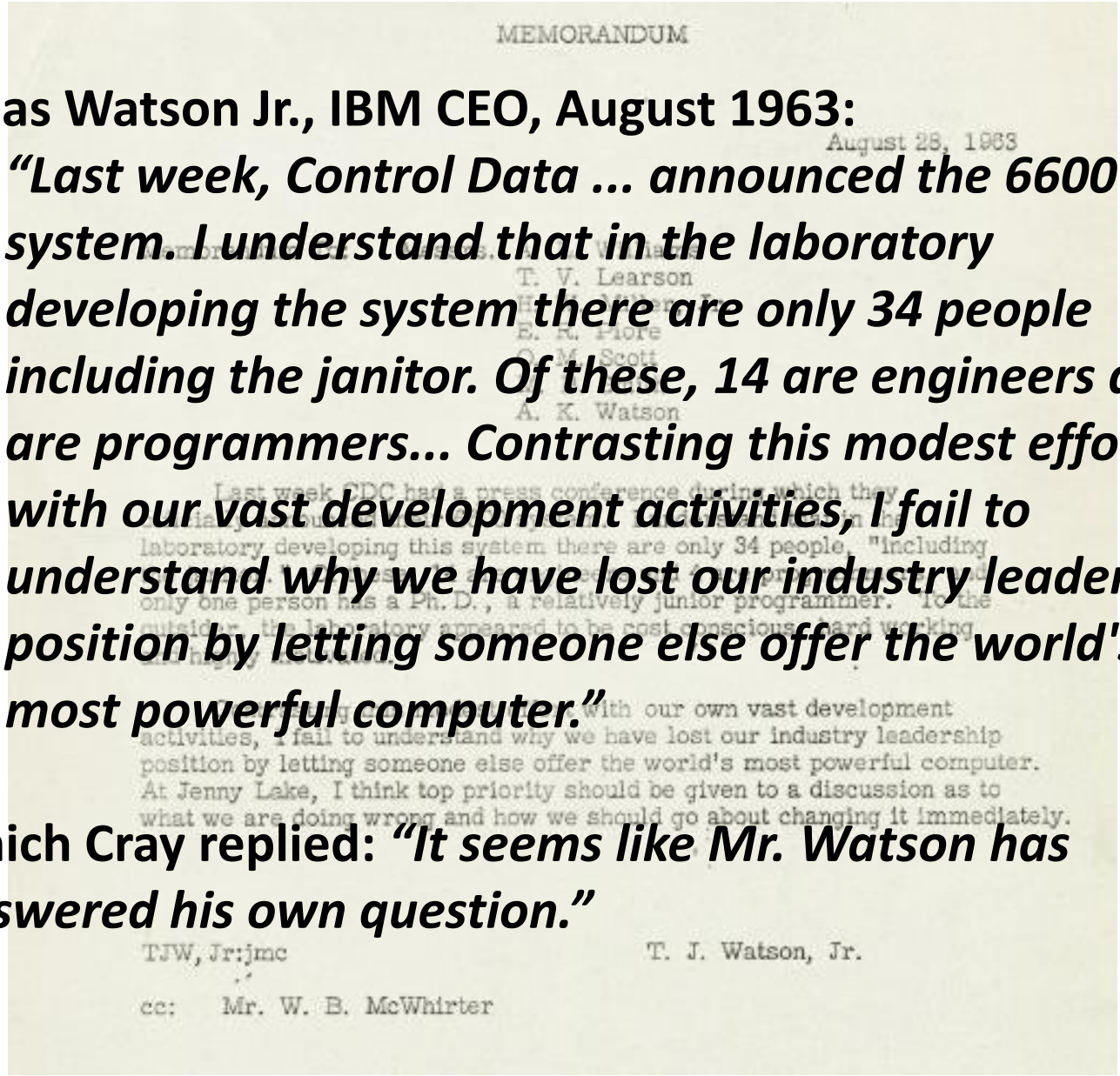
- **Goal of score-boarding** is to maintain an execution rate of CPI=1 by executing an instruction as early as possible
- **Instructions execute out-of-order** when there are sufficient resources and no data dependencies
- **A scoreboard is a hardware unit that keeps track of**
  - the **instructions** that are in the process of being executed
  - the **functional units** that are doing the executing
  - and the **registers** that will hold the results of those units
- **A scoreboard centrally performs** all hazard detection and instruction control

CDC 6000, *Seymour Cray, 1963*

Main features

- Ten functional units
- Scoreboard for dynamic scheduling of instructions
- Very fast clock, 10 MHz (FP add in 4 clocks)
- >400,000 transistors, 750 sq. ft., 5 tons, 150 kW,
- 3MIPS, fastest machine in world for 5 years (until CDC7600)
- over 100 sold ($6-10M each)

CDC 6000 *Seymour Cray, 1963*



MEMORANDUM

August 28, 1963

T. V. Learson

E. R. Fiore

C. M. Scott

A. K. Watson

Last week CDC had a press conference during which they announced the 6600 system. I understand that in the laboratory developing this system there are only 34 people, "including only one person has a Ph.D., a relatively junior programmer. To the outsider, the laboratory appeared to be cost conscious, hard working and highly motivated.

With our own vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer. At Jenny Lake, I think top priority should be given to a discussion as to what we are doing wrong and how we should go about changing it immediately.

TJW, Jr:jmc

T. J. Watson, Jr.

cc: Mr. W. B. McWhirter

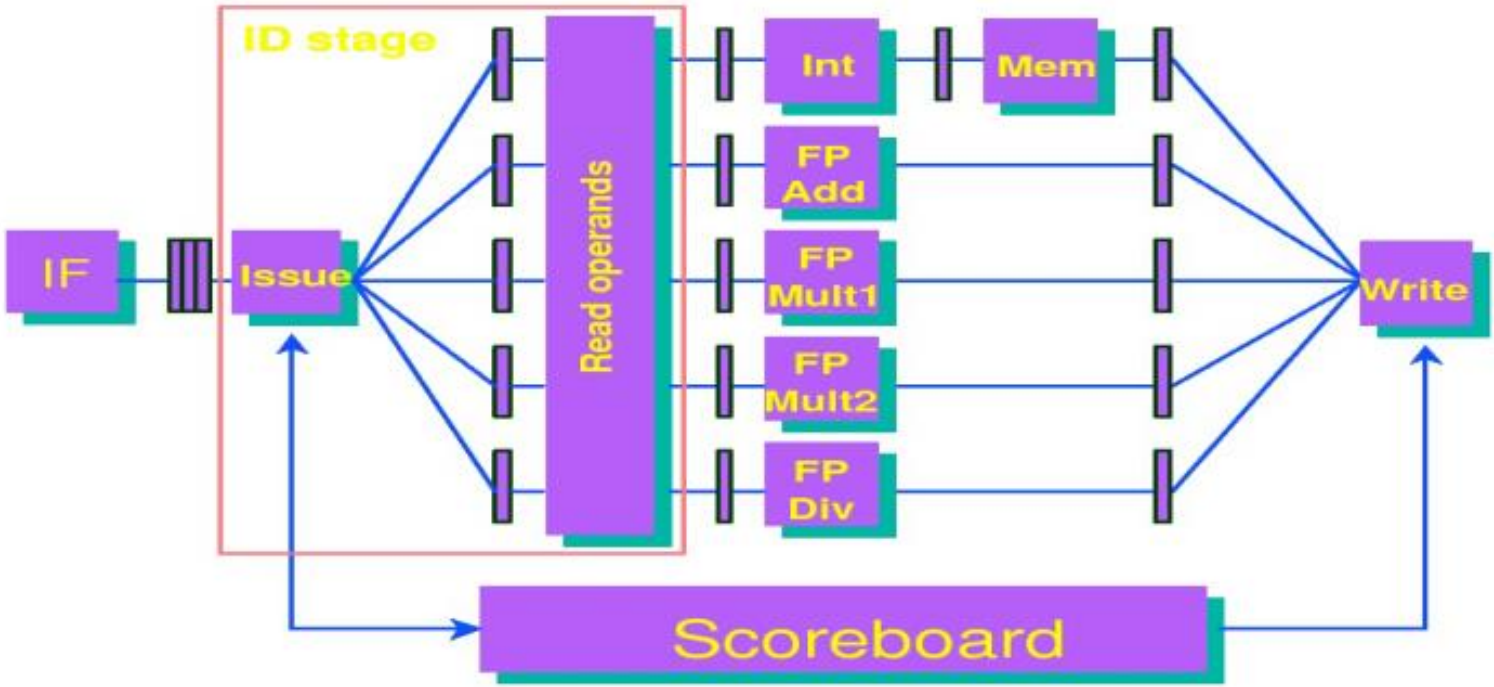
Thomas Watson Jr., IBM CEO, August 1963:

"Last week, Control Data ... announced the 6600 system. I understand that in the laboratory developing the system there are only 34 people including the janitor. Of these, 14 are engineers and 4 are programmers... Contrasting this modest effort with our vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world's most powerful computer."

To which Cray replied: *"It seems like Mr. Watson has answered his own question."*

Scoreboard pipeline

- **Issue:** decode and check for structural & WAW hazards
- **Read operands:** wait until no data hazards, then read operands
- **All data hazards are handled by the scoreboard**



Scoreboard complications

Out-of-order execution

⇒ WAR & WAW hazards

- **Solutions for WAR:**
  - Stall instruction in the **Write** stage until all previously issued instructions (with a WAR hazard) have read their operands
- **Solution for WAW:**
  - Stall in **Issue** until other instruction completes
- **RAW hazards handled in Read Operands**
- **Scoreboard keeps track of dependencies and state of operations**

Scoreboard functionality

- **Issue:** An instruction is issued if:
  - The needed functional unit is free (there is no **structural hazard**)
  - No functional unit has a destination operand equal to the destination of the instruction (resolves **WAW hazards**)
- **Read:** Wait until no data hazards, then read operands
  - Performed in parallel for all functional units
  - Resolves **RAW hazards** dynamically
- **EX:** Normal execution
  - Notify the scoreboard when ready
- **Write:** The instruction can update destination if:
  - All earlier instructions have read their operands (resolves **WAR hazards**)

Scoreboard components

- **Instruction status:** keeps track of which of the 4 steps the instruction is in
- **Functional unit status:** Indicates the state of the functional unit (FU). 9 fields for each FU:
  - ***Busy***: Indicates whether the unit is busy or not
  - ***Op***: Operation to perform in the unit (e.g. add or sub)
  - ***Fi***: Destination register name
  - ***Fj, Fk***: Source register names
  - ***Qj, Qk***: Name of functional unit producing regs Fj, Fk
  - ***Rj, Rk***: Flags indicating when Fj and Fk are ready
- **Register result status:** Indicates which functional unit will write each register, if any

Scoreboard example

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU								

Clock: 0

Detailed scoreboard control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rj(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

Scoreboard example, CP1

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				Integer				

Clock: 1

Scoreboard example, CP2

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Issue 2nd load?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Integer							

Clock: 2

Scoreboard example, CP3

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Issue MULT?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Integer							

Clock: 3

Scoreboard example, CP4

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				-				

Clock: 4

Scoreboard example, CP5

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU		Integer						

Clock: 5

Scoreboard example, CP6

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	Integer						

Clock: 6

Scoreboard example, CP7

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	Integer			Add			

Clock: 7

Scoreboard example, CP8

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4	-		Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		-	Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1	-			Add	Divide		

Clock: 8

Scoreboard example, CP9

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Read operands for MULT & SUB

Issue ADDD?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1				Add	Divide		

Clock: 9

Scoreboard example, CP11

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

SUBD completes execution

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1				Add	Divide		

Clock: 11

Scoreboard example, CP12

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Read operands for DIVD?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No							-	-
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1				-	Divide		

Clock: 12

Scoreboard example, CP13

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Issue ADDD

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			Add		Divide		

Clock: 13

Scoreboard example, CP16

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Can ADDD write result?

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			Add		Divide		

Clock: 16

Scoreboard example, CP17

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

ADDD stalls, waiting for DIVD to read F6

Resolves a WAR hazard!

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	Mult1			Add		Divide		

Clock: 17

Scoreboard example, CP20

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	-		Yes	Yes

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU	-			Add		Divide		

Clock: 20

Scoreboard example, CP21

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			No	No

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				Add		Divide		

Clock: 21

Scoreboard example, CP22

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

Now ADDD can safely write its result in F6

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
40	Divide	Yes	Div	F10	F0	F6			No	No

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU				-		Divide		

Clock: 22

Scoreboard example, CP62

Instruction status

Instruction		j	k	Issue	Read ops	Exec. compl.	Write result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Functional unit status

Time	Name	Busy	Op	dest Fi	src 1 Fj	src 2 Fk	FUsrc1 Qj	FUsrc2 Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	...	F30
FU						-		

Clock: 62

Factors that limits performance

The scoreboard technique is limited by:

- **The amount of parallelism available in code**
- **The number of scoreboard entries (window size)**
  - A large window can look ahead across more instructions
- **The number and types of functional units**
  - Contention for functional units leads to structural hazards
- **The presence of anti-dependencies and output dependencies**
  - These lead to WAR and WAW hazards that are handled by **stalling** the instruction in the Scoreboard
- **Number of data-paths to registers**